

Lab 5

Reg. No: 19BCE1209

Name: Gautam Sanjay Wadhwani

Course: CSE4001 Parallel and Distributed Computing

Q1. Sample critical without parallel program

Code:

```
#include<stdio.h>

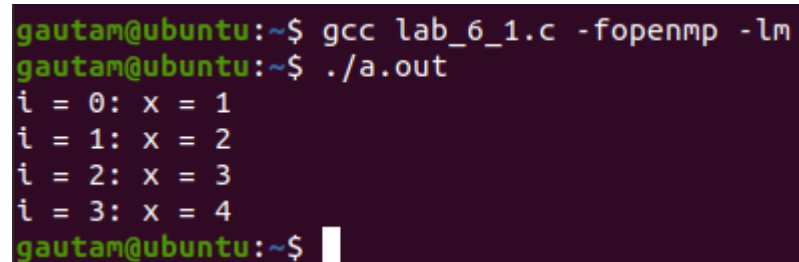
#include<stdlib.h>

#include<omp.h>

#include<math.h>

int main()
{
int n = 4, x = 0;
for(int i = 0; i < n; i++)
{
#pragma omp critical
x++;
printf("i = %d: x = %d\n", i, x);
}
return 0;
}
```

Output:



```
gautam@ubuntu:~$ gcc lab_6_1.c -fopenmp -lm
gautam@ubuntu:~$ ./a.out
i = 0: x = 1
i = 1: x = 2
i = 2: x = 3
i = 3: x = 4
gautam@ubuntu:~$
```

Q2. Sample static schedule program

Code:

```
#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

#include<math.h>

int main()

{

int n = 4, x = 0, i;

#pragma omp parallel for schedule(static)

for(i = 0; i < n; i++) {

    x++;

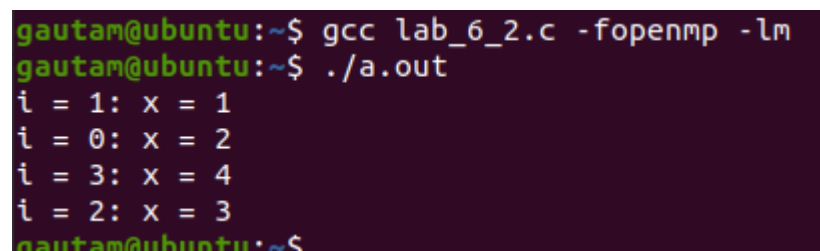
    printf("i = %d: x = %d\n", i, x);

}

return 0;

}
```

Output:



```
gautam@ubuntu:~$ gcc lab_6_2.c -fopenmp -lm
gautam@ubuntu:~$ ./a.out
i = 1: x = 1
i = 0: x = 2
i = 3: x = 4
i = 2: x = 3
gautam@ubuntu:~$
```

Q3. Sample dynamic schedule program

Code:

```
#include<stdio.h>

#include<stdlib.h>
```

```

#include<omp.h>

#include<math.h>


int main()
{
int n = 4, x = 0, i;

#pragma omp parallel for schedule(dynamic)
for(i = 0; i < n; i++) {

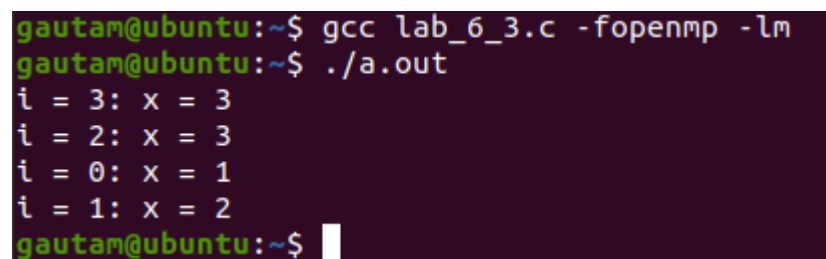
    x++;

    printf("i = %d: x = %d\n", i, x);
}

return 0;
}

```

Output:



```

gautam@ubuntu:~$ gcc lab_6_3.c -fopenmp -lm
gautam@ubuntu:~$ ./a.out
i = 3: x = 3
i = 2: x = 3
i = 0: x = 1
i = 1: x = 2
gautam@ubuntu:~$

```

Q4. Sample profile program with omp_get_wtime()

Code:

```

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

#include<math.h>


int main()
{

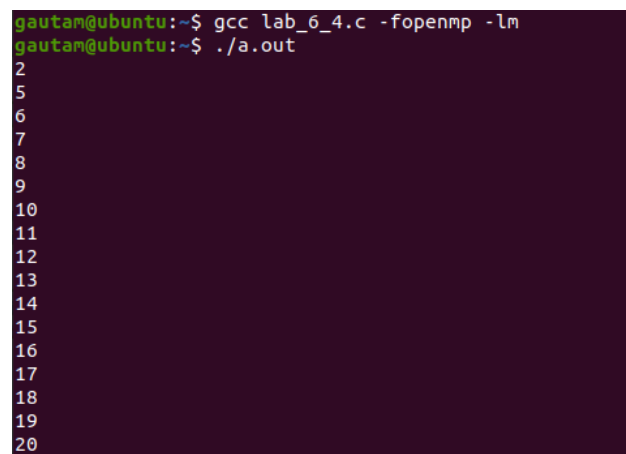
double start = omp_get_wtime(), end;

int x = 0;

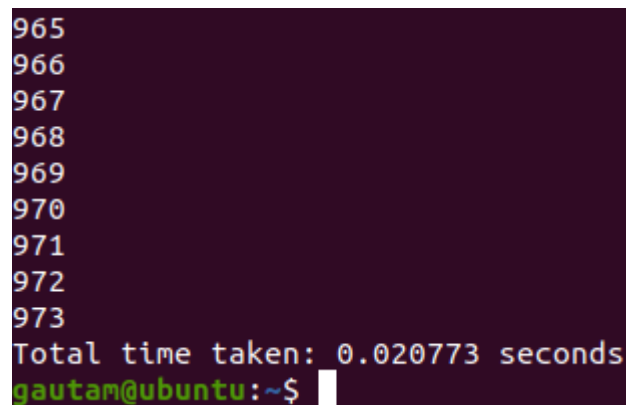
```

```
#pragma omp parallel for
for (int i = 0; i < 1000; i++)
{
    x++;
    printf("%d\n", x);
}
end = omp_get_wtime();
printf("Total time taken: %f seconds\n", end-start);
return 0;
}
```

Output:



```
gautam@ubuntu:~$ gcc lab_6_4.c -fopenmp -lm
gautam@ubuntu:~$ ./a.out
2
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```



```
965
966
967
968
969
970
971
972
973
Total time taken: 0.020773 seconds
gautam@ubuntu:~$
```

Q5. Sample program to set threads using directive clause

Code:

```
#include<stdio.h>

#include<stdlib.h>
```

```

#include<omp.h>

#include<math.h>

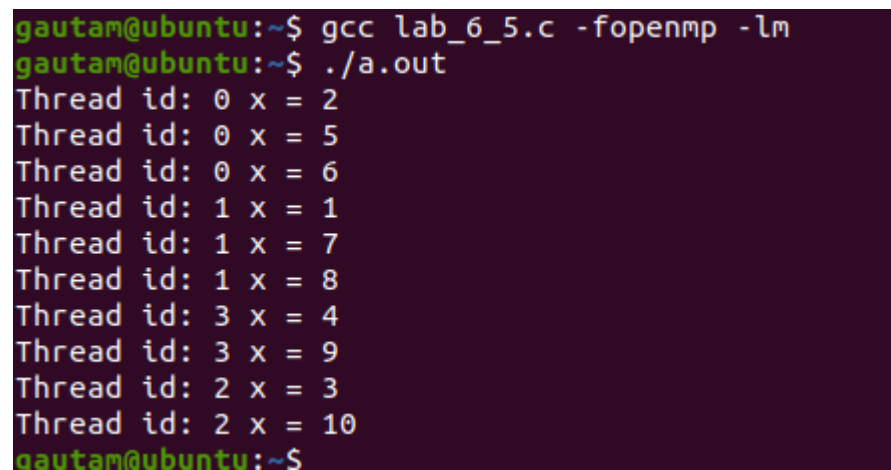

int main()
{
int x = 0;

#pragma omp parallel for num_threads(4)
for (int i = 0; i < 10; i++)
{
x++;

printf("Thread id: %d x = %d\n", omp_get_thread_num(), x);
}
return 0;
}

```

Output:



```

gautam@ubuntu:~$ gcc lab_6_5.c -fopenmp -lm
gautam@ubuntu:~$ ./a.out
Thread id: 0 x = 2
Thread id: 0 x = 5
Thread id: 0 x = 6
Thread id: 1 x = 1
Thread id: 1 x = 7
Thread id: 1 x = 8
Thread id: 3 x = 4
Thread id: 3 x = 9
Thread id: 2 x = 3
Thread id: 2 x = 10
gautam@ubuntu:~$

```

Q6. Sample program to set threads using `omp_set_threads()`

Code:

```

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

#include<math.h>

```

```

int main()
{
    omp_set_num_threads(4);

    int x = 0;

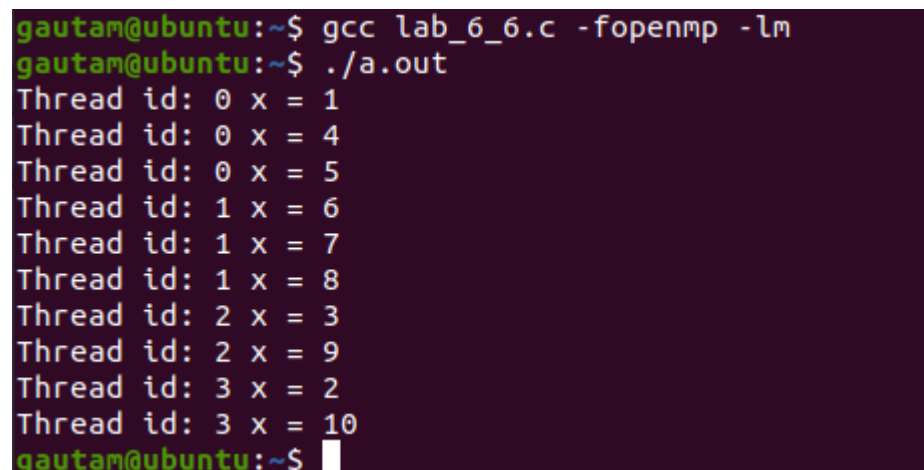
    #pragma omp parallel for
    for (int i = 0; i < 10; i++)
    {
        x++;

        printf("Thread id: %d x = %d\n", omp_get_thread_num(), x);
    }

    return 0;
}

```

Output:



```

gautam@ubuntu:~$ gcc lab_6_6.c -fopenmp -lm
gautam@ubuntu:~$ ./a.out
Thread id: 0 x = 1
Thread id: 0 x = 4
Thread id: 0 x = 5
Thread id: 1 x = 6
Thread id: 1 x = 7
Thread id: 1 x = 8
Thread id: 2 x = 3
Thread id: 2 x = 9
Thread id: 3 x = 2
Thread id: 3 x = 10
gautam@ubuntu:~$

```

Q7. Consider a priority queue that has four levels of priority. Each priority level can have at the maximum of 'n' tasks. Each task will have a priority. Depending on the priority, the tasks are inserted in the queue. For the dispatch of tasks, the tasks from the highest priority queue should be dispatched every clock cycle, from the second highest priority queue after every 2 clock cycles, from the third highest priority queue after every 3 clock cycles and so on. The insertion and dispatch can happen in parallel. Write an efficient OpenMP algorithm to emulate the above priority queue of an operating system. Every clock cycle, your program should print the following.

Code:

```

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

```

```

int main()
{
    int n;
    printf("Enter number of tasks: ");
    scanf("%d", &n);
    int tasks[n][2];
    int front[4] = {0, 0, 0, 0}, rear[4] = {0, 0, 0, 0};
    int q[4][1000];

    printf("Enter task id and priority for each task (higher number - higher priority, priority -> 1 to 4):\n");
    for(int i = 0; i < n; i++)
    {
        scanf("%d %d", &tasks[i][0], &tasks[i][1]);
        q[tasks[i][1]-1][rear[tasks[i][1]-1]++] = tasks[i][0]; // enqueue to the specified queue
    }
    for(int i = 1; i < 3*n + 1; i++)
    {
        #pragma omp parallel sections
        {
            #pragma omp section
            {
                if(front[3] < rear[3])
                {
                    printf("Dispatched task %d having priority 4 after %d clock cycles\n", q[3][front[3]], i);
                    #pragma omp critical
                    front[3]++;
                }
            }

            #pragma omp section
            {

```

```

if(i % 2 == 0 && front[2] < rear[2])
{
printf("Dispatched task %d having priority 3 after %d clock cycles\n", q[2][front[2]], i);
#pragma omp critical
front[2]++;
}
}

#pragma omp section
{
if(i % 3 == 0 && front[1] < rear[1])
{
printf("Dispatched task %d having priority 2 after %d clock cycles\n", q[1][front[1]], i);
#pragma omp critical
front[1]++;
}
}

#pragma omp section
{
if(i % 4 == 0 && front[0] < rear[0])
{
printf("Dispatched task %d having priority 1 after %d clock cycles\n", q[0][front[0]], i);
#pragma omp critical
front[0]++;
}
}

return 0;
}

```

Output:


```

gautam@ubuntu:~$ gcc lab_6_7.c -fopenmp
gautam@ubuntu:~$ ./a.out
Enter number of tasks: 8
Enter task id and priority for each task (higher number - higher priority, priority -> 1 to 4):
1 3
2 1
3 4
4 2
5 2
6 1
7 4
8 3
Dispatched task 3 having priority 4 after 1 clock cycles
Dispatched task 7 having priority 4 after 2 clock cycles
Dispatched task 1 having priority 3 after 2 clock cycles
Dispatched task 4 having priority 2 after 3 clock cycles
Dispatched task 8 having priority 3 after 4 clock cycles
Dispatched task 5 having priority 1 after 4 clock cycles
Dispatched task 5 having priority 2 after 6 clock cycles
Dispatched task 0 having priority 1 after 8 clock cycles
gautam@ubuntu:~$

```

Q8. Write a parallel program using OpenMP to implement the following series,
 $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots + \frac{1}{16384}$

Find the sum of the series and print it along with the thread id and the last value in the series i.e.,
 $(\frac{1}{16384})$.

Code:

```

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

#include<math.h>

int main()
{
    int n = 14, tid;

    double sum = 0;

    double x;

    #pragma omp parallel for reduction(+:sum) lastprivate(x, tid)
    for(int i = 1; i < n+1; i++)
    {
        x = 1.0/pow(2, i);

        sum += x;

        tid = omp_get_thread_num();

        printf("Current element: %f at i = %d calculated by thread %d\n", x, i, tid);
    }
}

```

```

printf("Total sum of series is %f\n", sum);

printf("Last element is %f, calculated by thread %d\n", x, tid);

return 0;

}

```

Output:

```

gautam@ubuntu:~$ gcc lab_6_8.c -fopenmp -lm
gautam@ubuntu:~$ ./a.out
Current element: 0.500000 at i = 1 calculated by thread 0
Current element: 0.250000 at i = 2 calculated by thread 0
Current element: 0.125000 at i = 3 calculated by thread 0
Current element: 0.062500 at i = 4 calculated by thread 0
Current element: 0.001953 at i = 9 calculated by thread 2
Current element: 0.000977 at i = 10 calculated by thread 2
Current element: 0.000488 at i = 11 calculated by thread 2
Current element: 0.031250 at i = 5 calculated by thread 1
Current element: 0.015625 at i = 6 calculated by thread 1
Current element: 0.007812 at i = 7 calculated by thread 1
Current element: 0.003906 at i = 8 calculated by thread 1
Current element: 0.000244 at i = 12 calculated by thread 3
Current element: 0.000122 at i = 13 calculated by thread 3
Current element: 0.000061 at i = 14 calculated by thread 3
Total sum of series is 0.999939
Last element is 0.000061, calculated by thread 3

```

Q9. There are four rail tracks that pass through a railway station. The trains can use the same track with a delay of 60 minutes. Depending on the above conditions, the signal for any of the track will be updated by a control system. The signal will be displayed red for time delay less than 60. If the time delay exceeds 60, the signal will be changed to green. The above tasks are done in parallel to avoid any mishaps. Implement the above scenario in C and parallelize it using OpenMP. Let the train arrival times are randomly generated in your program and number of trains that are going to be considered can be obtained from the user. The track to be used by a train can also be set by the user prior to the simulation. Whenever a signal is set, the same has to be displayed in your screen (Eg: Track 1: red, Track2 :Green etc) by your program.

Code:

```

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

int max(int a, int b)
{

```

```
if(a > b) return a;
else return b;
}
```

```
int main()
{
int n;
printf("Enter number of trains: ");
scanf("%d", &n);
int arrival_time[4][n], train_number[4][n], m[4] = {0, 0, 0, 0};
int signal[4] = {1, 1, 1, 1}; // 1 -> green, 0 -> red
int green_at[4] = {0, 0, 0, 0}; // make signal of track green at what time
printf("Enter train number, arrival time, track to be used each on new line in increasing order of
arrival time\n");
for(int i = 0; i < n; i++)
{
int train, arrival, track;
scanf("%d %d %d", &train, &arrival, &track);
arrival_time[track-1][m[track-1]] = arrival;
train_number[track-1][m[track-1]++] = train;
}

#pragma omp parallel for
for(int i = 0; i < 4; i++)
{
for(int j = 0; j < m[i]; j++)
{
printf("Track %d: Train %d time %d\n", i, train_number[i][j], max(green_at[i], arrival_time[i][j]));
green_at[i] = max(green_at[i], arrival_time[i][j] + 60);
}
}
}
```

```
return 0;
```

```
}
```

Output:

```
gautam@ubuntu:~$ gcc lab_6_9.c -fopenmp -lm
gautam@ubuntu:~$ ./a.out
Enter number of trains: 4
Enter train number, arrival time, track to be used each on new line in increasing order of arrival time
1 1 1
2 1 2
3 2 3
4 3 3
Track 0: Train 1 time 1
Track 1: Train 2 time 1
Track 2: Train 3 time 2
Track 2: Train 4 time 62
gautam@ubuntu:~$
```